(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 730 225 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
04.09.1996 Bulletin 1996/36

(51) Int. Cl.[6]: **G06F 9/38**

(21) Application number: **96103209.1**

(22) Date of filing: **01.03.1996**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: **03.03.1995 US 397893**

(71) Applicant: **HAL COMPUTER SYSTEMS, INC.**
Campbell, CA 95008 (US)

(72) Inventors:
- **Shebanow, Michael C.**
  Plano, Texas75075 (US)
- **Shen, Gene W.**
  Mountain View, CA 94043 (US)
- **Swami, Ravi**
  San Jose, CA 95130 (US)
- **Patkar, Niteen A.**
  Sunnyvale, CA 94087 (US)

(74) Representative: **Liesegang, Roland, Dr.-Ing.**
FORRESTER & BOEHMERT
Franz-Joseph-Strasse 38
80801 München (DE)

(54) Reclamation of processor resources in a data processor

(57) In a microprocessor, an apparatus is included for coordinating the use of physical registers in the microprocessor. Upon receiving an instruction, the coordination apparatus extracts source and destination logical registers from the instruction. For the destination logical register, the apparatus assigns a physical address to correspond to the logical register. In so doing, the apparatus stores the former relationship between the logical register and another physical register. Storing this former relationship allows the apparatus to backstep to a particular instruction when an execution exception is encountered. Also, the apparatus checks the instruction to determine whether it is a speculative branch instruction. If so, then the apparatus creates a checkpoint by storing selected state information. This checkpoint provides a reference point to which the processor may later backup if it is determined that a speculated branch was incorrectly predicted. Overall, the apparatus coordinates the use of physical registers in the processor in such a way that: (1) logical/physical register relationships are easily changeable; and (2) backup and backstep procedures are accommodated.

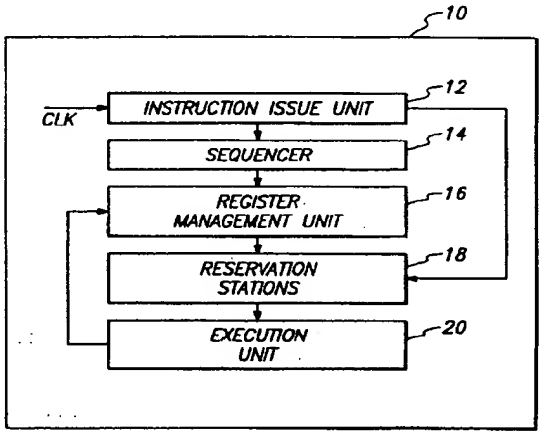

FIG. 1

EP 0 730 225 A2

Description

Related Applications

The subject matter of this application is related to the subject matter of the following applications:
European patent application 96101842.1;
European patent application 96101839.7;
European patent application 96101840.5;
European patent application 96101841.3;
the European patent application entitled "METHOD AND APPARATUS FOR ACCELERATING CONTROL TRANSFER RETURNS";
the European patent application entitled "METHOD AND APPARATUS FOR SELECTING INSTRUCTIONS FROM ONES READY TO EXECUTE";
the European patent application entitled "METHODS FOR UPDATING FETCH PROGRAM COUNTER";
the European patent application entitled "METHOD AND APPARATUS FOR RAPID EXECUTION OF CONTROL TRANSFER INSTRUCTIONS"
the European patent application entitled "ECC PROTECTED MEMORY ORGANIZATION WITH PIPELINED READ-MODIFY-WRITE ACCESSES":
the European patent application entitled "METHOD AND APPARATUS FOR PRIORITIZING AND HANDLING ERRORS IN A COMPUTER SYSTEM";
the European patent application entitled "HARDWARE SUPPORT FOR FAST SOFTWARE EMULATION OF UNIMPLEMENTED INSTRUCTIONS"; and
the European patent application entitled "METHOD AND APPARATUS FOR GENERATING A ZERO BIT STATUS FLAG IN A MICROPROCESSOR",
the latter eight of which are filed simultaneously with this application.

Field of the Invention

This invention relates generally to microprocessors, and more particularly to a method and apparatus for efficiently coordinating the use of physical registers in a microprocessor during instruction execution.

Background of the Invention

Recent improvements in data processing have included the processing of instructions in parallel. In order to implement parallel processing of instructions, various techniques have been implemented including register renaming, speculative execution, and out-of-order execution.

Register renaming is a technique utilized by processors in which the processor remaps the same architectural register to a different physical register in order to avoid stalling instruction issues. This technique requires the maintenance of a greater number of physical registers than would otherwise be warranted architecturally. The processor must, therefore, continuously monitor the status of the physical register resources including how many of the physical registers are in use at a given moment, to which architectural registers are the various physical registers mapped, and which of the physical registers are available for use. In order to accomplish this task, the processor maintains a list of physical registers ("freelist") that are not in use. When an instruction is issued, the processor remaps the architectural destination register to one of the registers on the freelist. The selected physical register is then removed from the freelist. Whenever the renamed physical registers are no longer needed, these physical registers are marked free by adding them to the freelist pool. Those physical register resources, which are missing from the freelist, are considered to be "in use" or otherwise unavailable to the processor for further mapping. Where the resultant register of an instruction is to be used as a source (architectural) register for a sequentially following instruction, the source register is mapped to a renamed physical register from the freelist. In order for the processor to use the correctly associated physical register, a rename map is continuously maintained by the processor which identifies which architectural registers are mapped to which physical registers. All sequentially subsequent instructions that refer to a sequentially earlier instruction's architectural register should use the renamed physical register.

Speculative execution is a technique utilized by processors in which the processor predicts a next branch target address for a next instruction where data is unavailable to evaluate a condition for a conditional branch. By using speculative execution, processor delays which would otherwise occur in waiting for the data needed to evaluate the condition, are avoided. Whenever there is a misprediction, the processor must be returned to the state which existed prior to the branching step and the correct branch must be identified in order to proceed with execution of the correct sequence of instructions. In order to recover the state of the processor after a misprediction, one technique that has been utilized is called checkpointing wherein the machine state is stored (or checkpointed) after every speculative instruction.

Out-of-order execution is a technique utilized by processors in which the processor includes multiple execution units which are issued instructions sequentially but which may complete execution of instructions non-sequentially due to varying execution times of the instructions.

In processors where the architectural registers are renamed, provisions must exist for efficiently restoring the correct state of the architectural registers when the processor does a backup to a checkpoint due to a mispredicted branch instruction, or, when a sequentially later instruction modifies the architectural register before the detection of an execution exception due to a sequentially earlier instruction.

To elaborate, many instruction sequences in a computer program are independent of other instruction

sequences. For example, in the following instruction stream,

```
1  Load LR1, Mem1;Load logical register LR1 with
                  data from Mem1
2  Inc LR1        ;Increment logical register LR1
3  Store LR1, Mem1;Store contents of logical regis-
                  ter LR1 into Mem1
4  Load LR2, Mem2;Load logical register LR2 with
                  data from Mem2
5  Inc LR2        ;Increment logical register LR2
6  Store LR2, Mem2;Store contents of logical regis-
                  ter LR2 into Mem2,
```

the second three instructions (instruction 4-6) are independent of the first three instructions (instruction 1-3). That is, instructions 4-6 do not depend on the results of instructions 1-3 to execute properly. Thus, in this example, instructions 1-3 and instructions 4-6 could be executed in parallel to optimize performance. It is this concept of executing instructions in parallel and out of sequence, which underlies the executing methods of superscalar processors.

To provide for parallel execution capability, superscalar processors typically comprise more physical registers than there are logical registers. Logical registers are registers, such as LR1 and LR2 in the example above, which are referenced in the instructions. Physical registers are the registers within the processor which are actually used for storing data during processing. The extra physical registers are needed in superscalar processors in order to accommodate parallel processing. One consequence of having more physical registers than logical registers is that there is no one-to-one correspondence between the logical and physical registers. Rather, a physical register may correspond to logical register LR1 for one set of instructions and then correspond to logical register LR2 for another set of instructions. Because the relationship between logical and physical registers can change, a mapping or coordination function is performed in order to keep track of the changing relationships. In order to optimize performance in a superscalar processor, an efficient method and apparatus for coordinating the use of the physical registers is needed.

Summary of the Invention

In accordance with the present invention, a data processor which executes multiple instructions in parallel uses renaming of physical registers in order to reduce delays in processing and uses checkpoints to retain the processor state and restore register resources on the occurrence of an instruction exception or branch misprediction. Additionally, the processor includes a resource reclamation random access memory (RAM) associating each issued instruction with an issue serial number (ISN) and storing information about the instruction including whether register renaming occurs, and what associated architectural and physical registers are utilized; and a resource reclamation pointer (RRP) for identifying instructions in program order that are prepared for retirement after successful execution and identifying any associated physical registers which may be added to the freelist.

The data processor includes a register file unit, a register reclaim file unit; a freelist unit, and a control unit, which coordinates the use of physical registers in a microprocessor in such a manner that allows for easy physical register assignment and convenient state restoration. The overall operation of the apparatus of the present invention is controlled by the control unit. In operation, the control unit receives an instruction, and in response, extracts a destination logical register value from the instruction. Then, the control unit obtains a free physical register identifier from the freelist unit which points to a particular physical register within the processor. Once the destination logical register value and the free physical register identifier are obtained, the control unit stores the logical register value into the register file unit in association with the obtained physical register identifier. By so doing, a relationship is established between the logical register value and the physical register identifier which can be used to map the logical register value to the particular physical register. A physical register is thus assigned or mapped to a logical register value.

In addition to establishing logical/physical register relationships, the apparatus of the present invention also performs at least two other important functions. First, if the instruction received is a branch instruction, then the apparatus creates a "checkpoint" which captures the current state of the processor. This checkpoint provides a reference point in time to which the processor can backtrack or backup if it is later determined that an incorrect branch was chosen. By creating these checkpoints, the apparatus of the present invention supports speculative execution. As a second important function, when the apparatus of the present invention assigns a new physical register to a logical register value, the old relationship between the logical register value and another physical register is saved in the register reclaim file unit. This is done so that if an execution exception (such as a divide-by-zero) is encountered which requires the execution of a trap handler, the old relationship can be easily restored. By so doing, the apparatus of the present invention provides the processor with a means for conveniently backstepping to a particular instruction before accessing a trap handler. Overall, the present invention provides a convenient and efficient method and apparatus for coordinating the use of physical registers in a microprocessor.

Brief Description of the Drawings

Figure 1 is a block diagram representation of a processor wherein the present invention is implemented.

Figure 2 is a detailed block diagram representation of the register management unit 16 of the present invention.

Figs. 3a-3c are block diagram representations of the register file unit 30, the register reclaim file unit 32, and the freelist unit 34 of the register management unit 16 in several sample states.

Figure 4 is an operational flow diagram of the control unit 36 of the control unit 36 of the register management unit 16.

Figure 5 is a more detailed flow diagram for the backup process 134 of Figure 4.

Figure 6 is a more detailed flow diagram for the backstep process 138 of Figure 4.

Detailed Description of the Preferred Embodiment

With reference to Figure 1, there is shown a block diagram representation of a processor 10 wherein the present invention is implemented. As shown, processor 10 preferably comprises an instruction issue unit 12, a sequencer 14 coupled to the instruction issue unit 12, a register management unit 16 coupled to the sequencer 14, reservation stations 18 coupled to both the register management unit 16 and the instruction issue unit 12, and an execution unit 20 coupled to the reservation stations 18 and the register management unit 16. In the preferred embodiment, processor 10 is a superscalar processor capable of executing multiple instructions in parallel.

In processor 10, the instruction issue unit 12 receives from an external source (not shown) a series of instructions, and stores these instructions for later execution. The instruction issue unit 12 receives a clock signal as input and in each clock cycle, unit 12 issues one or more instructions for execution. The issued instructions are sent to both the sequencer 14 and the reservation stations 18. The sequencer 14, in response to the issued instructions, assigns a sequence number (Sn) to each of the instructions. As will be explained later, these sequence numbers are used by the register management unit 16 to keep track of the instructions. Once the sequence numbers are assigned to the instructions, the sequencer 14 forwards the instructions to the register management unit 16.

Operationally, computer instructions reference logical registers. These logical registers (which will be referred to herein as LR) may be source registers which contain certain data needed to execute the instruction, or these registers may be destination registers to which data resulting from the execution of the instruction is to be written. For example, in the instruction

LR1 ÷ LR2 --> LR3,

which divides the contents of LR1 by the contents of LR2 and writes the result into LR3, LR1 and LR2 are the source registers while LR3 is the destination register.

Logical registers do not point to any physical location at which a physical register resides. To get from a logical register value to a physical register, a translation or mapping process is carried out. This mapping function is one of the functions performed by the register management unit 16. If there were a one-to-one correspondence between logical and physical registers, and if their relationships were constant, then the mapping function would be a simple one. All that would be needed is a static translation table. However, as noted previously, there is no constant one-to-one correspondence in superscalar processors. Instead, relationships between logical and physical registers are constantly changing. Hence, the register management unit 16 needs to be able to handle the changing relationships.

As an additional complication, superscalar processors engage in speculative execution. That is, whenever a branch instruction is encountered, a guess or prediction is made as to which branch will actually be taken. Once the prediction is made, the instructions following the predicted branch are executed. If later it is determined that the wrong branch was predicted, then the processor 10 will need to backtrack or "backup" to the branch instruction, choose the proper branch, and then execute instructions following that branch. In order to backup properly, the processor 10 will need to be restored to the state just prior to the branching operation. This involves, among other operations, restoring the relationships between the logical and physical registers. To accommodate this backup procedure, the register management unit 16 preferably coordinates the use of the physical registers in such a way that relationship restoration is possible.

As yet a further complication, superscalar processors process instructions in parallel and out of sequence. As a result, if an execution exception (such as a divide-by-zero) is encountered which requires the execution of a trap handler, it becomes necessary to "backstep" to the instruction invoking the exception before executing the trap handler. Like the backup procedure, this "backstep" procedure involves, among other operations, restoring the relationships between the logical and physical registers. Unlike the backup procedure, however, backstepping involves an instruction other than a branch instruction. This requires different handling, as will be explained below. The management unit 16 preferably manages the use of the physical registers in such a way that allows for this backstepping procedure. With the above background information in mind, the register management unit 16 will now be described in greater detail.

Figure 2 shows a detailed block diagram of the preferred embodiment of the register management unit 16. As shown, management unit 16 preferably comprises a register file unit 30, a register reclaim file unit 32, a freelist unit 34, and a control unit 36 for controlling the overall operation of the management unit 16.

The register file unit 30 is the component (or set of components) in management unit 16 which is mainly

responsible for mapping logical registers to physical registers, and for storing data. Register file unit 30 preferably comprises a number of different components, including a read-only-memory (ROM) 40, a content addressable memory (CAM) 42, a data random access memory (data RAM) 44, an address valid (AV) RAM 46, and at least one, and preferably a plurality of checkpoint RAM's 48_1-48_n. In register file unit 30, the ROM 40 is used to store all of the physical register address identifiers PR_0-PR_n for all of the physical registers in the processor 10. Each of the physical address identifiers PR_0-PR_n points to an entry in the data RAM 44 wherein data corresponding to the physical register identifier is stored, along with a data valid (DV) bit. Preferably, there is a one-to-one correspondence between the entries of the ROM 40 and the entries of the data RAM 44. Since the physical register identifiers PR_0-PR_n are stored in ROM 40, they cannot be altered by writing operations; hence, the physical register identifiers PR_0-PR_n remain constant for the life of the processor 10.

The CAM 42 in register file unit 30 is used to store logical register values corresponding to the physical register identifiers. Unlike ROM 40, the contents of the CAM 42 can be and are frequently changed: Preferably, there is a one-to-one correspondence between the entries of the CAM 42, the AV RAM 46, and the ROM 40, as shown in Figure 2. Together, ROM 40, CAM 42, and AV RAM 46 provide a mechanism for quickly and easily assigning a physical register to a logical register. To illustrate, suppose that it is desirable to assign physical register PR_0 to logical register LR_1. To establish such a relationship, all that needs to be done is to store the logical register value LR_1 into the entry of the CAM 42 corresponding to the physical register identifier PR_0, and to set the AV bit in AV RAM 46 corresponding to the physical register identifier PR_0. Once that is done, the next time the logical register LR_1 is asserted in an instruction, the CAM 42 will search for LR_1 and will signal a "hit" in the entry corresponding to the physical register identifier PR_0. This, in turn, will cause the identifier PR_0 to be outputted from the ROM 40, which in turn, will cause the corresponding entry in the data RAM 44 to be accessed. Thus, as shown by this example, storing a logical register value in CAM 42 establishes a direct relationship between a logical address and a physical address. To alter this relationship and to establish a new one, all that is needed is to store the logical register value LR_1 into a different entry in the CAM 42.

Register file unit 30 preferably further comprises checkpoint RAM's 48_1-48_n for temporarily storing the contents of the AV RAM 46. As will be explained in greater detail in a later section, whenever a speculative branch instruction is encountered, a "checkpoint" is made by storing the contents of the AV RAM 46 into one of the checkpoint RAM's 48_1-48_n. By storing the contents of the AV RAM 46, the state of the system prior to branching is captured. This checkpoint provides a reference point to backup to in case an incorrect branch is predicted. Register file unit 30 preferably comprises a plurality of checkpoint RAM's 48 in order to allow a plurality of checkpoints to be made. This in turn allows the processor 10 to execute through multiple levels of nested branches.

The register reclaim file unit 32 is the component in management unit 16 which makes it possible for the unit 16 to backstep to an instruction causing an execution exception. Preferably, unit 32 comprises a resource reclaim RAM 50 which has a plurality of entries. Each of the entries in the reclaim RAM 50 is preferably indexed with an issue sequence (or serial) number corresponding to a particular issued instruction. Associated with each entry is a rename valid bit portion, a first portion which stores a renamed architectural (logical) register identifier, and a second portion which stores an old physical register identifier. The rename valid bit portion stores a 'zero', if the particular instruction requires no renaming, and stores a 'one', if the particular instruction requires renaming of an architectural register. The old physical register identifier corresponds to the renamed architectural register identifier immediately before issue of the particular instruction. In effect, each entry of the reclaim RAM 50 stores a state of the system just prior to the execution of a particular instruction. Hence, the information stored in reclaim RAM 50 may be utilized to restore the system to the state that it had immediately prior to a given instruction. This aspect of the management unit 16 makes backstepping possible

Register management unit 16 preferably further comprises a freelist unit 34 which stores the physical register identifiers which are free to be assigned to logical registers. The freelist unit 34 preferably comprises a freelist RAM 62 which stores free physical address identifiers, a head register 64 which stores a "head" pointer, a tail register 66 which stores a "tail" pointer, and at least one, and preferably a plurality of checkpoint registers 68_1-68_n. The freelist RAM 62 is preferably operated as a FIFO storage. The head pointer points to the next physical register identifier in the RAM 62 which should be assigned to a logical register, while the tail pointer points to the last physical register identifier stored in the RAM 62. Each time a free physical register identifier is assigned to a logical register, the head pointer is incremented, and each time a free physical register identifier is added to the RAM 62, the tail pointer is incremented. Preferably, the freelist RAM 62 has at least (P-L) entries where P is the number of physical register identifiers and L is the number of logical register values.

With regard to the checkpoint registers 68_1-68_n, these are used to store the value of the head pointer whenever a speculative branch instruction is encountered. By storing the value of the head pointer, the state of the freelist unit 34 is saved. This in turn allows the state of the freelist RAM 62 to be restored if necessary. The checkpoint registers 48 provide support for the backup procedure. Preferably, freelist unit 34 comprises a plurality of checkpoint registers 48 to allow multiple checkpoints to be made. Multiple checkpoints enable

the processor 10 to execute through multiple levels of nested branches.

The register management unit 16 preferably further comprises a control unit 36 for coordinating the operation of the other components 30, 32, 34, and for controlling the overall operation of the management unit 16. In the preferred embodiment, the control unit 36 is implemented in hardware as a state machine. The control unit 36 will be described in greater detail as operation of the processor 10 is described.

Referring once again to Figure 1, the reservation stations 18 of the processor 10 are responsible for two primary functions. First, the reservation stations 18 capture all of the source data needed to execute an instruction. This data is received from the register management unit 16. Second, reservation stations 18 schedule instructions for execution. Instructions may be selected for execution if the DV bits from register file unit 30 are asserted for all sources and if no older instructions stored in the reservation stations are eligible. The instructions, once selected, are passed on to the execution unit 20 for execution. Overall, stations 18 are responsible for the smooth execution of instructions. All of the relevant elements of the processor 10 have now been discussed. The overall operation of the processor 10 will now be described.

Before the processor 10 is used in regular operation, it first needs to be initialized. Several steps are carried out in the initialization process. First, each and every logical register value is stored into one of the entries of the CAM 42. The AV bits corresponding to the CAM entries in which logical register values are stored are set. This ensures that before operation, all of the logical register values are validly mapped to a physical register. The particular mapping (i.e. which logical register value is mapped to which physical register) is arbitrary. No single logical register value is mapped to more than one physical register, however.

Once all of the logical register values are stored into the CAM 42, the free physical register are known. Accordingly, the physical register identifiers corresponding to the free registers are stored into the freelist RAM 62. This provides an indication as to which physical registers are free and may be assigned.

As an example, suppose that the management unit 16 is initialized as shown in Figure 3a. More specifically, suppose that physical register PR_0 has been assigned to logical register LR_0, PR_1 has been assigned to LR_1, PR_2 has been assigned to LR_2, PR_3 has been assigned to LR_3, and PR_4 has been assigned to LR_4. After assignment, data ($Data_0$-$Data_4$) may be written into the corresponding locations in the data RAM 44, along with asserted data valid bits. As shown in Figure 3a, physical registers PR_5-PR_9 have not been assigned to any logical register value. Thus, they are considered "free", which means that they may be assigned logical register values in upcoming operations. Thus, the physical register identifiers PR_5-PR_9 associated with the free physical registers are stored in the freelist unit 34. The head pointer in the freelist unit is pointing to PR_5, thereby indicating that PR_5 will be the next physical register assigned to a logical register value. Currently, no information is stored in the register reclaim file unit 32.

Now, suppose that the instruction issue unit 12 issues the following instruction:

$$LR_0 \div LR_1 \text{ --> } LR_3.$$

This instruction, when executed, will cause the data in logical registers LR_0 to be divided by the data in logical register LR_1, and will cause the result to be stored into logical register LR_3. For this instruction, logical registers LR_0 and LR_1 are the source logical registers from which data will be drawn, while logical register LR_3 is the destination logical register. Once issued, the instruction is passed on to the sequencer 14, where a sequence number is assigned to the instruction. Suppose that sequence number 0 is assigned. Thus, the instruction becomes:

$$\text{Instruction \#0: } LR_0\ LR_1 \text{ --> } LR_3.$$

Once a sequence number is assigned, the instruction is passed on to the control unit 36 of the register management unit 16 for processing.

In the management unit 16, it is the control unit 36 which receives and processes new instructions. An operational flow diagram for the control unit 36 is shown in Figure 4. Preferably, control unit 36 begins operation by receiving 100 the instruction and then determining 102 whether the instruction is a speculative branch instruction. If the instruction is a speculative branch instruction, then control unit 36 preferably creates a "checkpoint" to provide a reference point to which the processor 10 can backup in case the wrong branch of the branch instruction is predicted. In creating a checkpoint, two operations are taken. First, the contents of the AV RAM 46 are stored 104 into one of the checkpoint RAM's 48_1-48_n. This operation preserves for later reference all of the current relationships between the logical registers and the physical registers. Second, the contents of the head counter 64 in the freelist unit 34 are stored 106 into one of the checkpoint registers 68_1-68_n. By storing these two sets of information, the current state of the processor 10 is recorded. This information may be retrieved at a later time to restore the state of the processor 10 to that just prior to the execution of the speculative branch instruction. As will be explained in greater detail later, this aspect of the management unit 16 enables the processor 10 to carry out the backup procedure of the present invention.

In the present example, the instruction ($LR_0 \div LR_1$ --> LR_3) is not a speculative branch instruction; thus, control unit 36 bypasses steps 104 and 106 and proceeds to step 108 to extract a destination logical register value from the instruction. In the present example, the destination logical register value is LR_3. Once the destination logical register value is extracted, control

unit 36 accesses the freelist RAM 62 to extract 110 the next available physical register identifier therefrom. As shown in Figure 3a, the head pointer is currently pointing to physical register identifier PR_5; thus, PR_5 is selected as the physical register to assign to the logical register LR_3. Thereafter, control unit 36 increments 112 the head counter 64 to cause the counter to point to the next available free physical register, which in the example is PR_6.

After a physical register identifier is retrieved from the freelist unit 34, control unit 36 applies 114 the logical register value LR_3 to the CAM 42. In effect, this operation checks for the current physical register assignment for logical register LR_3. In the present example, LR_3 is currently assigned to physical register PR_3, as shown in the register file unit 30. Thus, when LR_3 is applied to the CAM 42, a hit will be found which will cause the physical register identifier PR_3 to be read out of the ROM 40. This physical register identifier PR_3, along with the logical register value LR_3, is then stored into the reclaim RAM 50 in the entry corresponding to the sequence number of the current instruction (sequence #0), as shown in Figure 3b. The information in reclaim RAM 50 will be used, if necessary, to backstep the processor 10 at a later time to the current state of the processor 10.

After the logical register value LR_3 and its corresponding old physical register identifier PR_3 are stored into the reclaim RAM 50, control unit 36 clears the AV bit corresponding to the old physical register assignment, which in the example is the AV bit in AV RAM 46 corresponding to the physical register identifier PR_3. Once that is done, control unit 36 loads 120 the logical register value LR_3 into the CAM 42 in the entry corresponding to the new physical register identifier, which is PR_5. The logical register value LR_3 is thus assigned to a new physical register. Thereafter, the AV bit corresponding to physical register PR_5 is set 122 to indicate that this is now the current physical register assignment for logical register LR_3. Note, however, that the data valid DV bit corresponding to PR_5 is not set. This is because no data has yet been written into the corresponding entry of the data RAM 44 because the instruction has not yet been executed. The DV bit will be set once the instruction is executed and appropriate data is stored into the proper entry of the data RAM 44.

Note from Figure 3b that at this point, the CAM 42 has two entries wherein logical register value LR_3 is stored. This would appear to cause confusion. However, note that only the AV bit corresponding to the current physical register assignment (PR_5) is set, thereby indicating that the entry corresponding to PR_5 is the current assignment. The AV bit corresponding to the old physical register (PR_3) is not set. This manipulation of the AV bit forestalls any confusion that might arise due to multiple instances of the same logical register value. Using the process thus far described, a destination logical register may be assigned to a new and different physical register.

The assignment of a new physical register to a destination logical register is only one of the functions performed by control unit 36. The other function is to retrieve the source data needed to execute the instruction. Steps 124-130 of Figure 4 illustrate this retrieval process. In the present example, the source logical registers are LR_0 and LR_1. Hence, in order to execute the instruction, data will need to be retrieved from the storage registers indicated by LR_0 and LR_1.

In carrying out the retrieval process, control unit 36 first determines 124 whether any source logical registers are indicated by the instruction. If source logical registers are indicated by the instruction, as is the case in the present example, then control unit 36 begins the retrieval process by applying 126 the source logical registers LR_0, LR_1 to the CAM 42. Specifically, when LR_0 is applied to CAM 42, a hit is found in the entry of the CAM 42 corresponding to physical register identifier PR_0. Since the AV bit for this entry is set to "1", this hit causes the physical register identifier PR_0 to be outputted from the ROM 40, which in turn, causes the corresponding data (Dat_0) to be outputted from the data RAM 44. The data from logical register LR_0 is thereafter sent 128 to the reservation stations 18. Since the data valid DV bit for this entry is set to "1", the data will be used by the reservation stations in executing the instruction. A similar process takes place when LR_1 is applied to the CAM 42. Specifically, the application of LR_1 causes a hit to be found in the entry of the CAM 42 corresponding to physical register identifier PR_1. This hit causes the physical register identifier PR_1 to be outputted from ROM 40. This in turn causes the data ($Data_1$) corresponding to PR_1 to be outputted from the data RAM 44 to the reservation stations. Data from logical register LR_1 is thus passed on to the reservation stations. Again, since the data valid bit corresponding to the outputted data is set, the reservation stations 18 will use the data in executing the instruction.

After data from the source logical registers LR_0, LR_1 are sent to the reservation stations 18, control unit 36 further sends 130 the physical register identifier assigned to the destination logical register. In the present invention, the destination logical register is LR_3 and the physical register assigned to LR_3 is PR_5. Hence, in step 130, control unit 36 sends physical address identifier PR_5 to the reservation stations. After step 130, the reservation stations 18 have all of the information needed to execute the instruction. Thus, stations 18 schedule the instruction for execution, and at the appropriate time, sends the instruction along with the information discussed above to the execution unit 20. In response, execution unit 20 executes the instruction and writes the resulting data into an appropriate entry in the data RAM 44. Since the physical register identifier PR_5 was sent to the execution unit 20, the resulting data will be written into the entry of the data RAM 44 corresponding to the physical register identifier PR_5, which is the correct entry. In addition, execution unit 20 preferably sets the data valid bit corresponding

to the entry to indicate that the data in the entry is now valid and can be used as source data. Thereafter, execution of the instruction is complete.

The process described above is what takes place when an instruction executes smoothly. However, as noted previously, two events may occur which may affect smooth execution. The first event is an acknowledgment that an incorrect branch was taken at a speculative branch instruction. The second event is the invocation of an execution exception which requires a trap handler to be executed. To remedy the first event, a backup procedure is implemented to restore the processor 10 to the state it had just prior to the branch instruction. To remedy the second event, a backstep procedure is implemented to restore the processor 10 to the state it had just prior to the instruction invoking the execution exception. In the event that an incorrect branch was taken, the sequencer 14 will issue a "backup" control signal to the control unit 36 of the register management unit 16. In the event of an execution exception, the execution unit 20 will issue a "backstep" control signal to the control unit 36. In steps 132 and 136, control unit 36 checks for these control signals. If one of these control signals is detected, then control unit 36 will take appropriate action.

To illustrate the backup procedure, suppose that a speculative branch instruction is issued and sent to the register management unit 16. In processing this instruction, control unit 36 first determines 102 (Figure 4) whether the instruction is a speculative branch instruction. If the instruction is a speculative branch instruction, as it is in the present example, then control unit 36 proceeds to carry out steps 104 and 106 to create a "checkpoint". In step 104, the contents of the AV RAM 46 are stored into one of the checkpoint RAM's 48_1, as shown in Figure 3c. In step 106, the contents of the head counter 64 in the freelist unit 34 are stored into one of the checkpoint registers 68_1. These two operations preserve the state of the processor 10 prior to execution of the branch instruction to create a reference state to which the processor may return. Once that is done, the instruction is processed in the same manner as other instructions.

Suppose now that during the following instruction or several instructions thereafter it is discovered that a speculative branch was mis-predicted and that a wrong branch was taken. In such a case, it will be necessary to restore the processor 10 to the state that it had just prior to the branch instruction. This restoration or backup is achieved as follows. First, the sequence 14 generates and sends a "backup" signal to the control unit 36. This "backup" signal is detected by control unit 36 in step 132 and in response, control unit 36 implements the backup procedure shown in Figure 5. Preferably, control unit 36 begins the backup procedure by overwriting 152 the contents of the AV RAM 46 with the contents of the checkpoint RAM 48_1 (Figure 3c) which were stored in the checkpoint RAM 48a during step 104. Further, control unit 36 overwrites 154 the contents of the head counter 64 in the freelist unit 34 with the contents of the checkpoint register 68_1 which were stored in the checkpoint register 68_1 in step 106. By carrying out these two steps, control unit 36 restores the processor 10 to the state that it had just prior to the branch. Processor backup is thus achieved.

It is sometimes necessary to restore a machine state not only to a checkpointed location but to an instruction which lies between checkpoints. An example of such a situation is one where an instruction encounters a divide-by-zero condition. Since a divide-by-zero is not possible, such an instruction usually invokes an exception trap. Before accessing the trap handler, however, it is first necessary to backstep the machine state to that which existed immediately after the instruction was executed. This "backstep" is usually difficult to achieve because there is no checkpoint created for the instruction since the instruction is not a branch instruction. With the present invention, however, backstepping to an instruction can be performed easily.

To illustrate this backstep procedure, reference will be made to Figs. 3b, 4, and 6, and to a specific example. To draw upon a previously used example, suppose again that the instruction

$$\text{Instruction \#0: } LR_0 \div LR_1 \text{ --> } LR_3$$

is received by the control unit 36 of the register management unit 16. Suppose further again: (1) that the logical register value LR_3 is written into the CAM entry corresponding to the physical identifier entry PR_5 as shown in Figure 3b, thereby assigning LR_3 to physical register PR_5; and (2) that the logical register value LR_3 and its corresponding former physical register identifier PR_3 are stored into the reclaim RAM 50 in the entry indexed by the instruction sequence #0. The writing of these values into CAM 42 and reclaim RAM 50 were described previously with reference to steps 108-122 of Figure 4. Hence, these operations need not be re-described here. The important points to note here are: (1) that there are two instances of LR_3 stored in the CAM 42, one corresponding to the currently assigned physical register identifier PR_5, and one corresponding to the formerly assigned physical register identifier PR_3; and (2) that the reclaim RAM 50 stores the former relationship between LR_3 and PR_3. It is the information in reclaim RAM 50 which allows for easy backstepping.

To illustrate how backstepping is achieved, suppose that the data corresponding to LR_1 is a zero. If such is the case, then the above instruction would be a divide-by-zero operation. Upon executing this instruction, the execution unit 20 will issue a "backstep" control signal. This signal is detected by control unit 36 in step 136, and in response, control unit 36 carries out the backstep procedure 138 of the present invention. The backstep procedure is shown in greater detail in Figure 6.

Preferably, control unit 36 begins the backstep procedure by determining 162 the sequence number associated with the instruction which caused the execution

exception. In the present example, the sequence number associated with the instruction is 0. Once determined, the instruction sequence number is used as an index to retrieve 164 from the reclaim RAM 50 the logical register value LR_3 and its corresponding former physical register identifier PR_3. Thereafter, control unit 36 applies the logical register value LR_3 retrieved from the reclaim RAM 50 to the CAM 42 to determine 166 which physical register is currently assigned to the logical register LR_3. As shown in Figure 3b, physical register PR_5 is currently assigned to LR_3. Once the current assignment is found, control unit 36 clears 168 the AV bit corresponding to the currently assigned physical register. Hence, in the present example, the AV bit corresponding to PR_5 is cleared. Thereafter, control unit 36 writes 170 the logical register value LR_3 into the CAM 42 at an entry corresponding to the former physical address identifier. Hence, in the present example, LR_3 is written into the CAM entry corresponding to the physical register identifier PR_3. Once that is done, control unit 36 sets 172 the AV bit corresponding to the former physical address identifier PR_3. By carrying out the above steps, the current logical register/physical register relationship is erased and the former relationship is reinstated. As a final step, the head counter 64 in the freelist unit 34 is decremented 174 so that it once again points to PR_5 as it did before the above instruction was processed. Thus, the state of the processor 10 is restored or backstepped to the desired state. The trap handler may now be accessed.

At this point, it should be noted that the backstep procedure is quite similar to the register assignment procedure discussed in steps 108-122 of Figure 4. The only difference is that instead of assigning the logical register value to a free physical register, the logical register value is assigned back to a physical register with which it had a previous relationship. The similarity between the two processes is significant because it means that the same hardware used to implement the register assignment process can be used to implement the backstep procedure. Thus, very little additional hardware (in fact, a simple additional multiplexer) is required to implement the backstep procedure.

The backstepping procedure has been described as backstepping over only one instruction. It should be noted, however, that the method and apparatus of the present invention may be used to backstep over any desired number of instructions. Typically, however, the number of instructions which may be backstepped over is limited to the number of instructions which can be renamed per cycle. This number may vary from system to system.

Referring again to Figure 4, after control unit 36 carries out steps 100-138, it makes a determination as to whether to send a certain physical register identifier back to the freelist unit to make the corresponding physical register available again for reassignment. A physical register identifier will be sent or released to the freelist unit 34 if the instruction for which the physical register is used is committed. An instruction will be committed if the instruction successfully completed execution and if all previous instructions completed execution without error or exceptions. In step 140, control unit 36 makes a determination as to whether a certain instruction has been committed. If so, then control unit 36 releases the physical register identifier or identifiers associated with the instruction to the freelist unit 34. Continuing with the present example, suppose that the instruction

$$LR_0 \div LR_1 \text{ --> } LR_3$$

is committed. In such a case, the physical register identifier PR_3 (stored in the reclaim RAM 50) corresponding to the destination register LR_3 is passed 142 to the freelist unit 34. More specifically, control unit 36 adds the physical register identifier PR_3 to the tail of the freelist RAM 62 and then increments the tail counter 66. The free register PR_3 is thus added to the list of physical registers which may be assigned to a logical register.

The present invention has been described with reference to specific examples. However, the invention should not be construed to be so limited. Various modifications may be made by one of ordinary skill in the art with the benefit of this disclosure without departing from the spirit of the invention. For example, for the sake of simplicity, the invention has been described with the assumption that only one instruction is issued per cycle. It should be noted, though, that the invention may be and is actually intended to be implemented in processors which issue multiple instructions per clock cycle. To accommodate multiple instructions per cycle, more ports may be added to the register file unit 30, the register reclaim file unit 32, and the freelist unit 34. These and other modifications are within the spirit and contemplation of the invention. Therefore, the present invention should not be limited by the examples used to illustrate it but only by the scope of the appended claims.

Claims

1. A method for executing multiple, sequential instructions in parallel within a data processing device comprising the steps of:

   issuing a series of sequential instructions including a first and second instruction;

   identifying a first logical register with a physical register in accordance with the first instruction;

   identifying a second logical register with the physical register in accordance with the second instruction;

   executing the second instruction out of sequence with the first instruction,

   retaining device state information including an identification of the sequential ordering of the second instruction, the identification of the second logical register with the physical register, and the identification of the first logical register with the physical register; and,

restoring the device state information on the occurrence of a data corrupting event.

2. A method for issuing each of a sequence of instructions utilized within a parallel processing device comprising the steps of:

identifying the sequential order of an instruction;

determining whether a logical register is referenced by the instruction and if a logical register is referenced:

associating the referenced logical register with a physical register, and

determining whether the referenced logical register was associated with a prior physical register from a previously issued instruction; and

storing state information including:

an identifier of the sequential ordering of the instruction,

the association of the referenced logical register with the physical register, if the instruction references a logical register, and

the association of the referenced logical register with the prior physical register, if the logical register was previously associated with a prior physical register.

3. The method as in Claim 2, wherein the storing step includes

storing a rename identifier identifying a change of physical register association.

4. A method for multiple instruction execution of a sequence of instructions within a parallel processing device comprising the steps of:

issuing each of a sequence of instructions by

identifying the sequential order of an instruction;

determining whether a logical register is referenced by the instruction and if a logical register is referenced:

associating the referenced logical register with a physical register, and

determining whether the referenced logical register was associated with a prior physical register from a previously issued instruction; and

storing state information including:

an identifier of the sequential ordering of the instruction,

the association of the referenced logical register with the physical register, if the instruction references a logical register, and

the association of the referenced logical register with the prior physical register, if the logical register was previously associated with a prior physical register;

executing the instruction sequence by

simultaneously selecting multiple instructions.

5. The method of Claim 4, including the step of:

upon the occurrence of a data corrupting event, stopping execution of selected instructions; and

returning the state of the processing device to a point in time prior to corrupting the data by

backstepping the state in reverse sequence by utilizing the previously stored state information for each issued instruction executed after the point in time, and avoiding changing the state if the rename identifier does not identify a change of physical register association for a selected instruction.

6. The method of Claim 4, including the step of:

reclaiming physical registers by

monitoring execution completion of each instruction in the sequence;

upon execution completion of an instruction in the sequence:

determining whether any other instructions in the sequence which are earlier in sequential order have not completed execution, and

if no earlier instructions have not completed execution, retrieving the previously stored association of the prior referenced physical register and designating availability of the prior referenced physical register.

7. A method for executing multiple, sequential instructions in parallel within a data processing device comprising the steps of:

issuing each of a sequence of instructions including a first and second instruction by;

identifying the sequential order of an instruction;

determining whether a logical register is referenced by the instruction and if a logical register is referenced:

associating the referenced logical register with a physical register, and

determining whether the referenced logical register was associated with a prior physical register from a previously issued instruction; and

storing state information including:

an identifier of the sequential ordering of the instruction,

the association of the referenced logical register with the physical register, if the instruction references a logical register, and

the association of the referenced logical register with the prior physical register, if the logical register was previously associated with a prior physical register;

identifying a first logical register with a physical register in accordance with the first instruction;

identifying a second logical register with the physical register in accordance with the second instruction;

executing the second instruction out of sequence with the first instruction; and,

restoring the device state information on the occurrence of a data corrupting event.

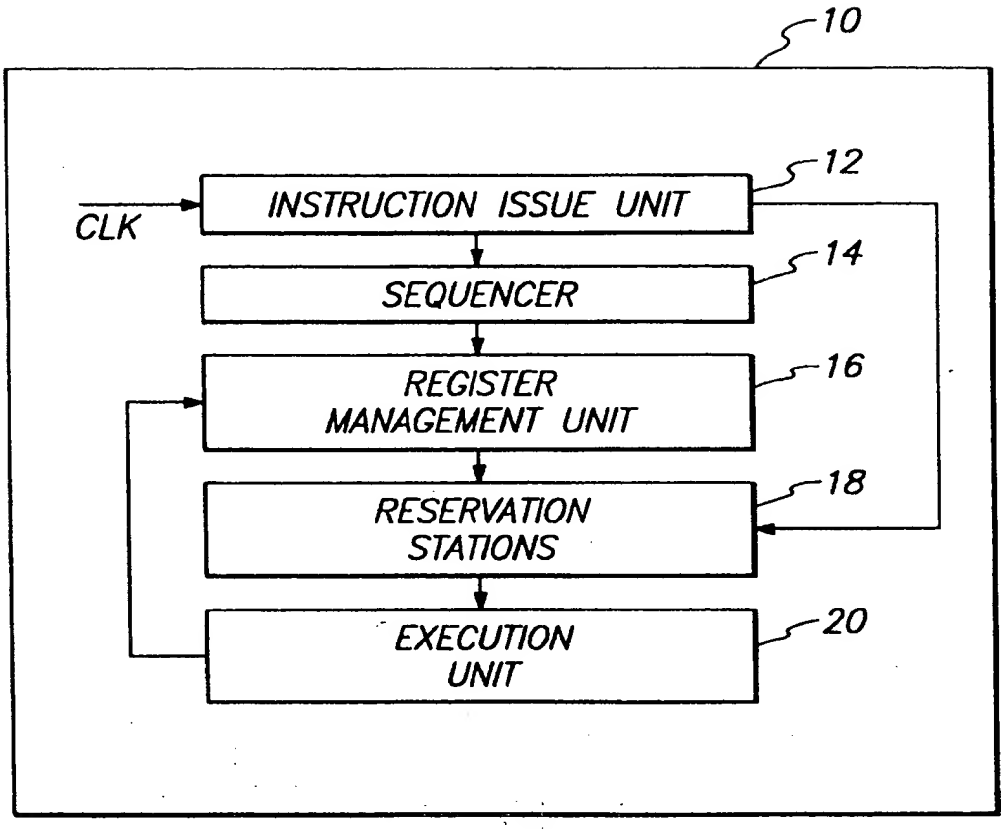

FIG. 1

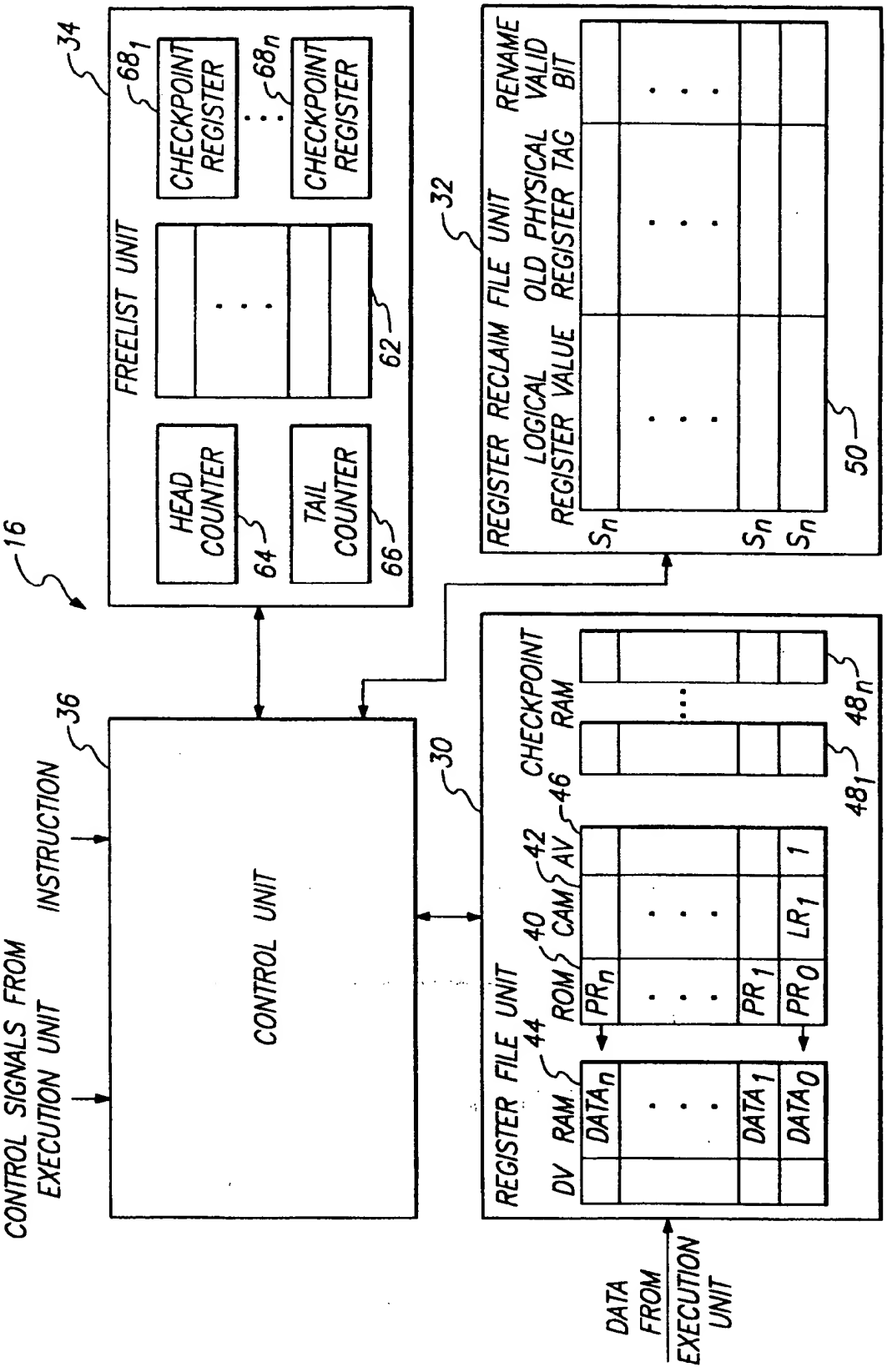

FIG. 2

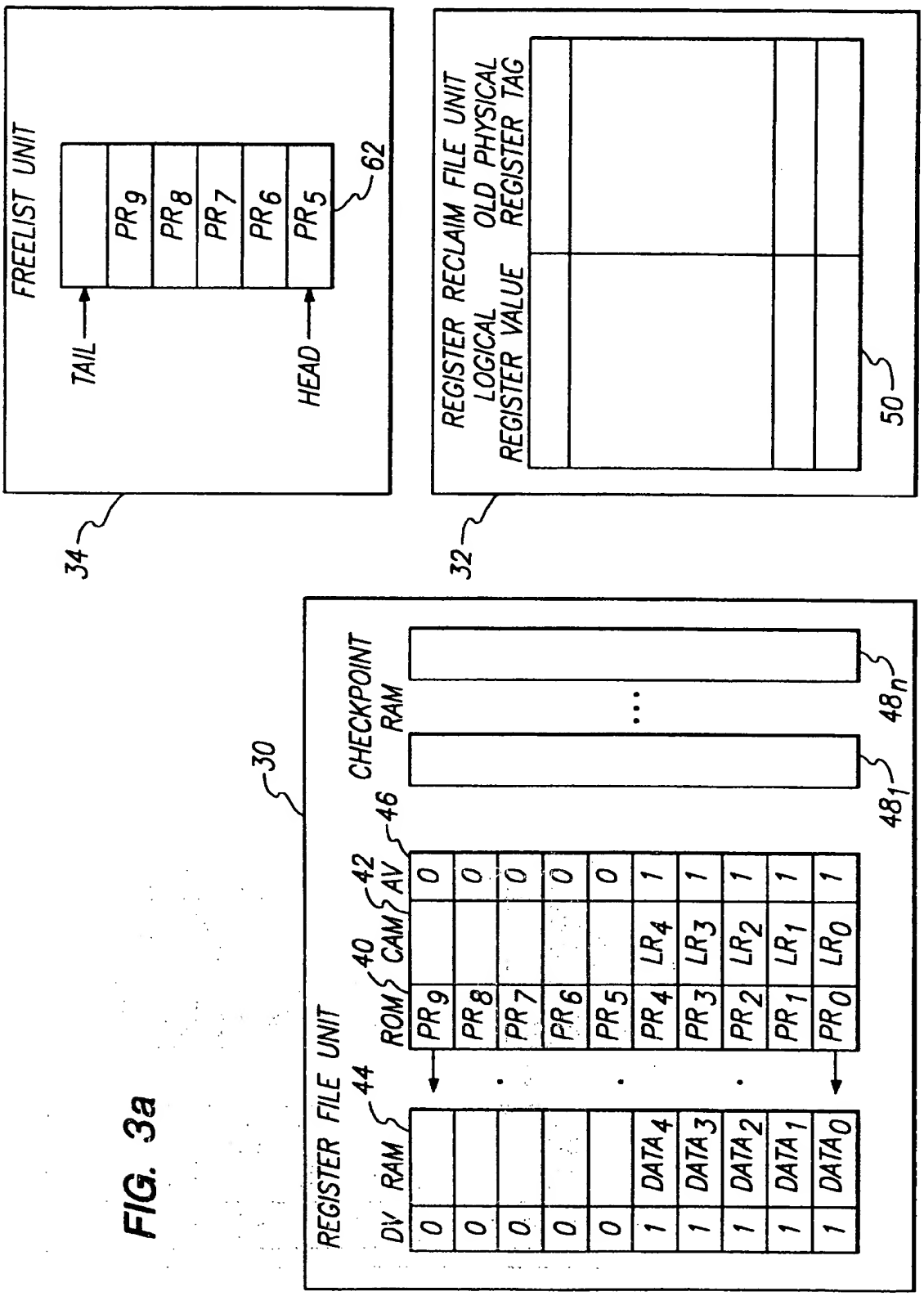

FIG. 3a

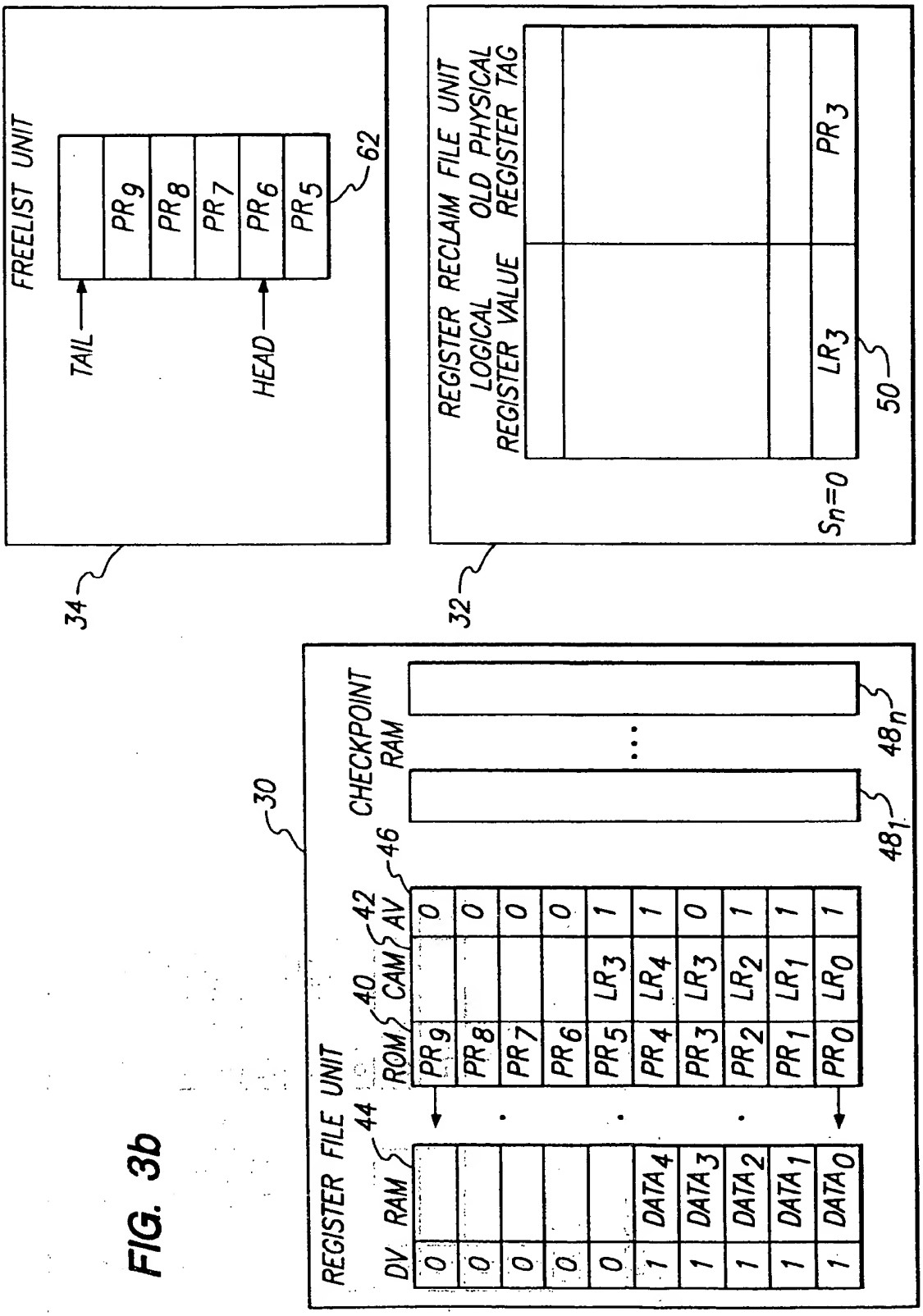

FIG. 3b

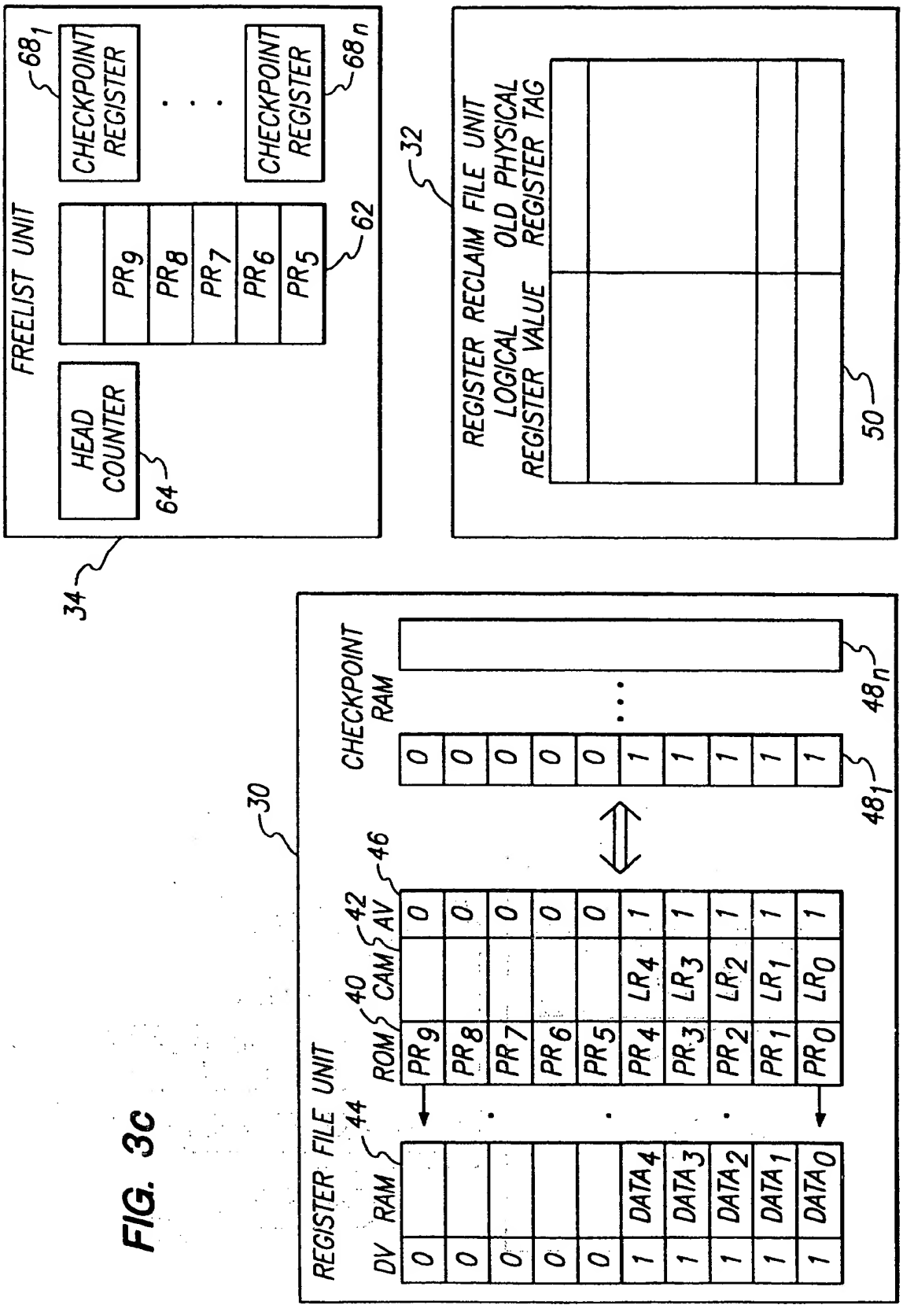

FIG. 3c

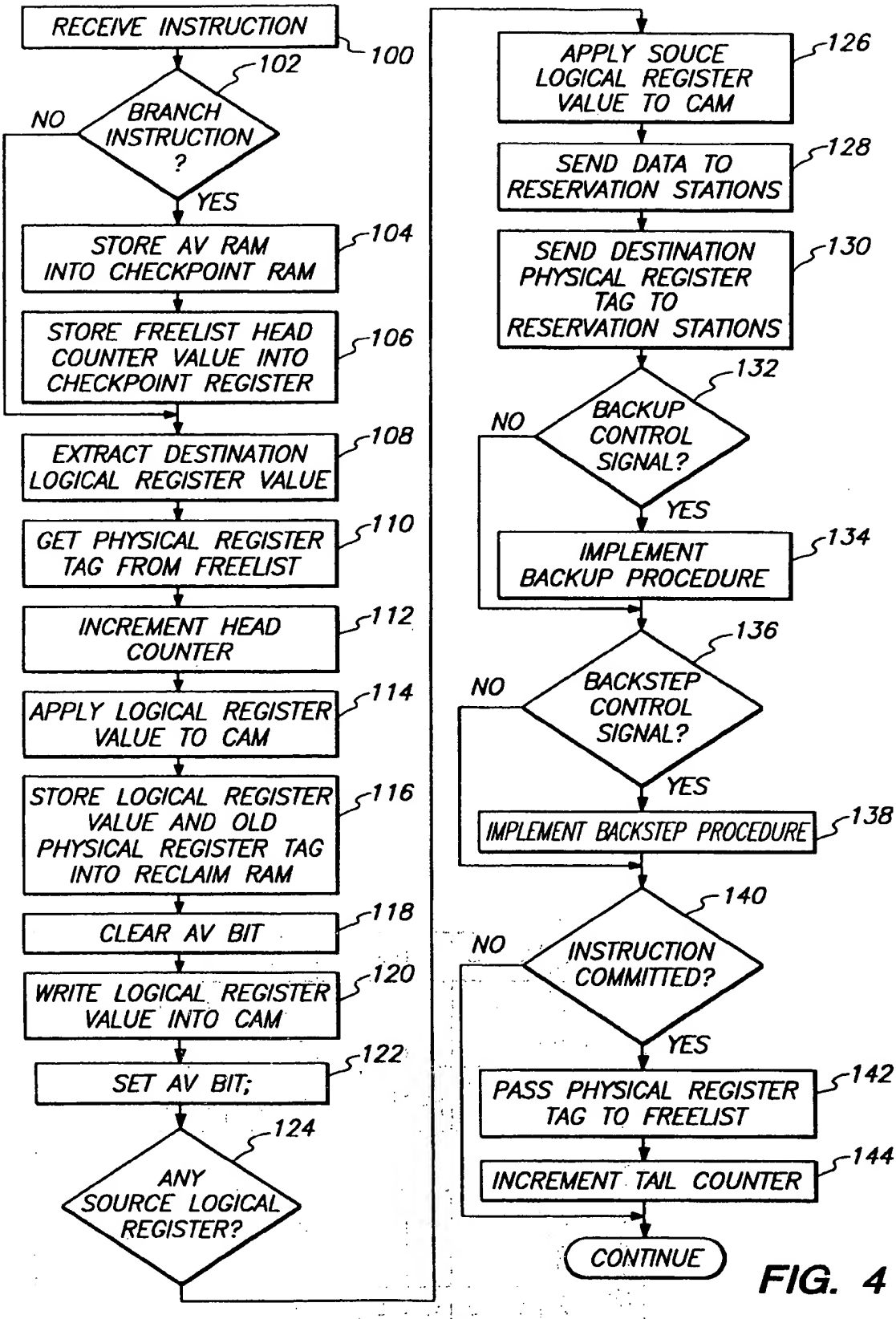

FIG. 4

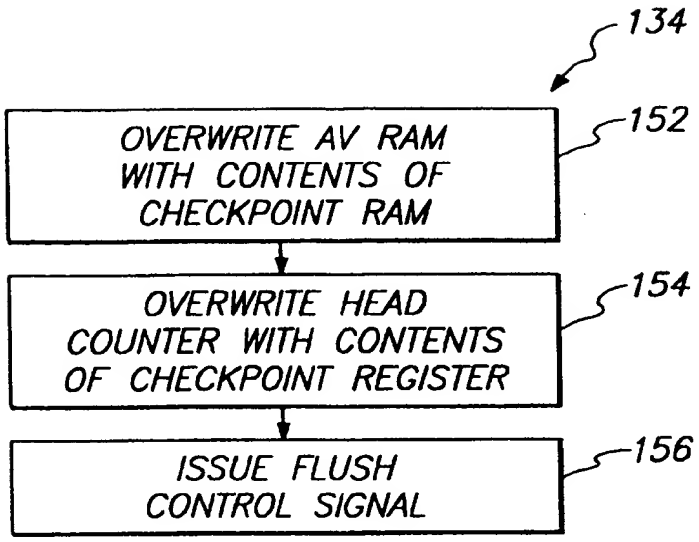

FIG. 5

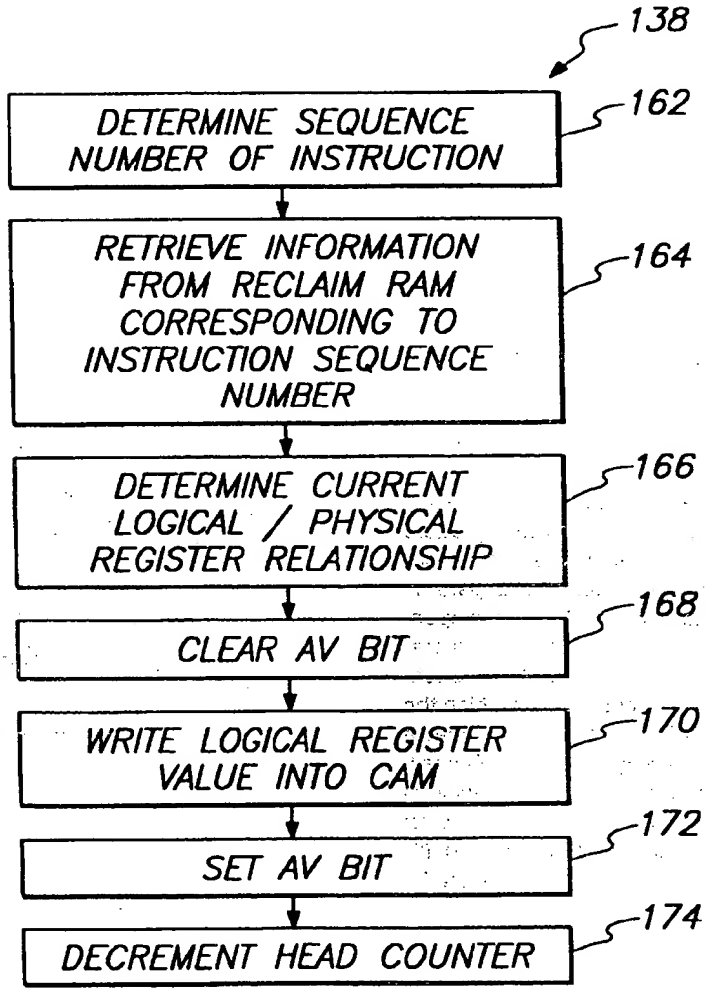

FIG. 6

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 730 225 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
08.12.1999 Bulletin 1999/49

(43) Date of publication A2:
04.09.1996 Bulletin 1996/36

(21) Application number: **96103209.1**

(22) Date of filing: **01.03.1996**

(51) Int. Cl.[6]: **G06F 9/38**

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priority: **03.03.1995 US 397893**

(71) Applicant:
HAL COMPUTER SYSTEMS, INC.
Campbell, CA 95008 (US)

(72) Inventors:
- **Shebanow, Michael C.**
  Plano, Texas75075 (US)
- **Shen, Gene W.**
  Mountain View, CA 94043 (US)
- **Swami, Ravi**
  San Jose, CA 95130 (US)
- **Patkar, Niteen A.**
  Sunnyvale, CA 94087 (US)

(74) Representative:
Liesegang, Roland, Dr.-Ing.
FORRESTER & BOEHMERT
Franz-Joseph-Strasse 38
80801 München (DE)

(54) Reclamation of processor resources in a data processor

(57) In a microprocessor, an apparatus is included for coordinating the use of physical registers in the microprocessor. Upon receiving an instruction, the coordination apparatus extracts source and destination logical registers from the instruction. For the destination logical register, the apparatus assigns a physical address to correspond to the logical register. In so doing, the apparatus stores the former relationship between the logical register and another physical register. Storing this former relationship allows the apparatus to backstep to a particular instruction when an execution exception is encountered. Also, the apparatus checks the instruction to determine whether it is a speculative branch instruction. If so, then the apparatus creates a checkpoint by storing selected state information. This checkpoint provides a reference point to which the processor may later backup if it is determined that a speculated branch was incorrectly predicted. Overall, the apparatus coordinates the use of physical registers in the processor in such a way that: (1) logical/physical register relationships are easily changeable; and (2) backup and backstep procedures are accommodated.

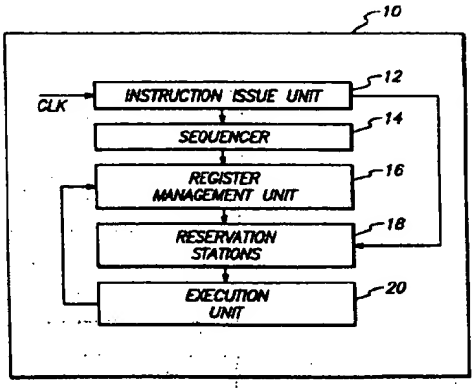

FIG. 1

EP 0 730 225 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 10 3209

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	"CONDITIONAL EXECUTION IN A REGISTER MANAGEMENT SCHEME FOR OUT OF SEQUENCE EXECUTION" IBM TECHNICAL DISCLOSURE BULLETIN, vol. 34, no. 10A, 1 March 1992 (1992-03-01), pages 449-454, XP000256730 ISSN: 0018-8689	1-4,6,7	G06F9/38
A	* the whole document *	5	
A	EP 0 515 166 A (MOTOROLA INC) 25 November 1992 (1992-11-25) * the whole document *	1-7	
A	EP 0 600 611 A (IBM) 8 June 1994 (1994-06-08) * page 8, line 6 - page 9, line 25 *	1-7	
A	EP 0 463 628 A (DIGITAL EQUIPMENT CORP) 2 January 1992 (1992-01-02)		

TECHNICAL FIELDS SEARCHED (Int.Cl.6)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	13 October 1999	Daskalakis, T

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 96 10 3209

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

13-10-1999

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0515166 A	25-11-1992	US 5355457 A	11-10-1994
		JP 5216663 A	27-08-1993
EP 0600611 A	08-06-1994	CA 2098414 A,C	01-05-1994
		CN 1105138 A	12-07-1995
		JP 2698033 B	19-01-1998
		JP 6214784 A	05-08-1994
		KR 9704509 B	28-03-1997
		US 5481683 A	02-01-1996
EP 0463628 A	02-01-1992	US 5197132 A	23-03-1993
		AU 624043 B	28-05-1992
		AU 7527891 A	12-03-1992
		CA 2042767 A	30-12-1991
		JP 2037700 C	28-03-1996
		JP 6044071 A	18-02-1994
		JP 7072865 B	02-08-1995

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82